Int. J. on Recent Trends in Engineering and Technology, Vol. 8, No. 1, Jan 2013

A Kernel-Level Traffic Probe to Capture and Analyze Data Flows with Priorities

Luis Zabala[1], Alberto Pineda[1], Armando Ferro[1], and Lander Alonso[1]
[1] University of the Basque Country, Bilbao, Spain
Email: {luis.zabala, alberto.pineda, armando.ferro}@ehu.es, lalonso005@ikasle.ehu.es

***Abstract*—This paper describes the proposal of a priority flow oriented design of the Ksensor architecture. Ksensor is a multiprocessor traffic capture and analysis system for high speed networks developed at kernel space. While the current architecture permits the capture and analysis of data flows, there are several scenarios where it does not perform adequately to achieve this goal, for example, if a certain type of traffic is more valuable than others. Thus, this work pursues the design that allows Ksensor to provide data flow treatment to a larger extent. This improvement will allow the new architecture to provide more reliability in data flow capture and processing.**

***Index Terms*—data flow monitoring; priorities; high-speed networks; kernel-level traffic probe; Ksensor**

I. Introduction

The variety of services offered via the Internet as well as advances in networking technologies have made passive monitoring applications play an increasingly important role for network administrators. Intrusion Detection Systems (IDS), Quality of Service (QoS) measurement systems and traffic monitoring systems are examples of this type of applications. The first requirement of these systems is to capture data in an efficient way. However, an acceptable high-speed packet capture is not enough, unless it is accompanied by appropriate filtering and analysis capabilities. Because of the current data transmission rates (10 Gbps and above), this issue cannot be solved easily, especially when using general purpose operating systems running over commodity off-the-shelf hardware [1].Networking, Quality and Security (NQaS) research group has developed a kernel-level multiprocessor traffic probe that captures and analyzes network traffic in high speed networks [2]. This probe, called Ksensor, improves the capture performance of a previous architecture that worked at user space [3]. Ksensor is implemented as a modification of the Linux Kernel.

Ksensor captures as many packets as possible using the resources of the system efficiently but it does not tell packets from different data flows apart. However, in many applications, such as QoS measurements, traffic flow detection and session tracking are needed. Taking all this into account, this work presents the development of a solution that enhances the features of Ksensor, adding ability to filter data flows according to some established priorities. There are several works which improve the performance of the packet capturing and filtering, for example [4][5]. However, none of them introduces the enhancement by modifying the behavior of the Linux networking subsystem's software interrupt (softIRQ). This is exactly the basis of the proposal presented in this paper, as will be seen below.The rest of the paper is organized as follows. In Section II we explain the architecture of Ksensor and its problems in traffic flow analysis. Section III describes the new design proposed for the traffic probe. In Section IV, the modules included in the design are detailed. Section V deals with the validation and comparative tests. Finally, Section VI remarks the conclusions of the paper.

II. Scope of Work

In the previous work [2], our research group proposed a design for a probe, called Ksensor, able to cope with high-speed traffic monitoring using commodity hardware. The design of Ksensor is based on execution threads defined to take advantage of multiprocessor architectures.

A. Ksensor Packet Capture System

Ksensor's packet capture system is based on a modification of the Linux Kernel network subsystem. It can be seen in Fig. 1. When a packet arrives to the network interface card (NIC), it is stored in the memory of the card. Then, it is copied to a buffer using Direct Memory Access (DMA). This means that the CPU is not involved in the operation. Finally, the softIRQ is responsible for the capturing, by moving the packets from the buffer to the packet queue [2] of Ksensor.

The system generates a processing thread per CPU. Each thread takes packets from the packet queue and analyzes them. However, only one of the processors can run the capture task [6]. Therefore, one of the system processors has to divide its resources into capture and analysis tasks, while the rest only perform analysis tasks

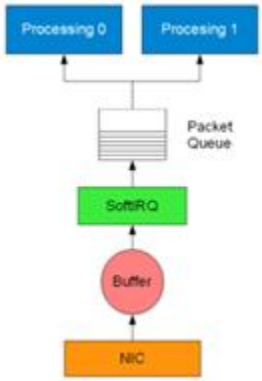

Fig. 1. Ksensor packet capture system.

DOI: 01.IJRTET.8.1.65

If the traffic rate is high, Ksensor may not be able to analyze all the packets. In cases like that, packets received when the packet queue is full are rejected. However, Ksensor provides a congestion avoidance mechanism [2] that can be used in livelock situations for disabling the packet capture. This way, all the resources are devoted to the analysis tasks.

B. Ksensor Packet Analysis Phase

The packet analysis phase of Ksensor is performed by one processing thread per CPU (Processing 0 and 1 in Fig. 1). For that purpose, Ksensor has its own memory map, which is created in a dedicated segment of shared memory [7]. This memory is used to store the logic that threads use to process packets. It does not matter which thread analyzes which packet, even if the packets are from different sessions, because all the threads share that memory segment [8].

C. Ksensor's Limitations in Data Flow Processing with Priorities

Some network traffic analyses focus on the treatment of the whole data flow. However, in many applications like QoS measurement systems and IDS, traffic probes have to do session tracking. In order to do that, the probe has to distinguish different traffic flows. In these cases, the system is required to capture every packet flow of interest or, at least, the most of them.

In livelock situations, the system does not check the flows which the rejected packets belong to. There is no way to ensure the capture of packets from the most relevant flows rejecting the rest. Therefore the rejection of packets from high priority rated flows can reduce analysis performance, since more flows are being processed with fewer packets on each. Furthermore, the analysis of some flows could be impossible if most of their packets are not captured. Even, there may be situations in which packets from more relevant flows are lost and some packets from not so relevant flows are processed. In order to prevent those situations, we have to add new functionalities to Ksensor.

III. Framework Design

As stated before, the Ksensor architecture has to be extended to work in scenarios requiring traffic flow sensitive analysis. To achieve this goal, the architecture needs to integrate two new functions. On the one hand, we need a system that rejects packets selectively, prioritizing the more relevant flows. On the other hand, we need a performance controller system. Depending on the state of Ksensor, this controller system configures the different resources and policies in order to process the packets according to the priority of their flows.

Fig. 2 shows the proposed solution which integrates new modules in Ksensor's architecture. These modules will only come into operation when the new execution mode, called "Flow_prio", is activated.Packet filtering is done by modifying Ksensor's capturing stage, with the insertion of the module known as PDS (Packet Discarding System).Regarding the control system, FPCS (Flow_prio Performance Controller

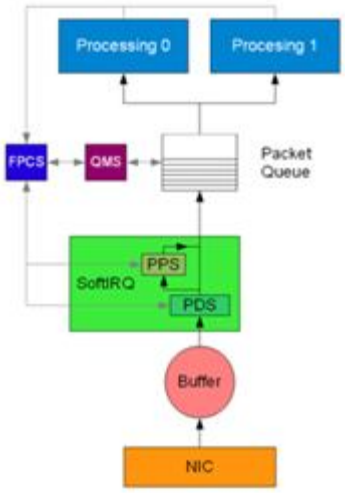

Fig. 2. Architecture to process data flows with priorities in Ksensor.

System) carries out the functions associated with it. These two modules, as well as the rest appearing in Fig. 2, will be explained in more detail in Section IV.This approach allows discarding packets at a lower level, thus preventing unwanted traffic to fill the queue once the FPCS has decided that packets belonging to certain flows should be dropped.

However, the integration of new modules can impact on Ksensor's performance negatively. Since the proposal involves modifying the code of the softIRQ, this approach can have impact on the performance of the packet capturing. For this reason, the implementation of the new features should have the least possible impact from a computational point of view. In a pre-livelock situation, the CPUs are not operating at full capacity which allows integrating new functionalities without affecting performance. In a livelock situation, however, introducing functionalities will consume resources decreasing the system's performance but it will ensure that the captured and analyzed traffic is the desired one.

IV. Modules and Policies of the New Architecture

This section describes the modules added by the design of this proposal, which allow Ksensor to make the processing of data flows with priorities.

A. Packet Discarding System (PDS)

This is a packet filtering system which compares every captured packet with the fixed filtering policy, before any analysis. According to the policy rules, the PDS decides to enquee the packet for analysis, to pass it to the Packet Preprocessing System (PPS) or to drop it. Hence, this system drops the packets that the administrator of the system decides not to process. In this way, the PDS allows not to waste computing power to perform tasks for packets that are not analyzed later. Several filters can be implemented in the PDS, depending on protocol, TCP/UDP port, IP address, etc.As can be seen in Fig. 2, the PDS is run within the softIRQ. In general, the softIRQ is a critical system routine [9]. If the load of the softIRQ is increased, it may affect the overall

DOI: 01.IJRTET.8.1.65

performance of the system. Thus, because this execution point is delicate, the computational impact of PDS must be as low as possible.

B. Packet Discarding Policy (PDP)

The PDP is the decision logic that the PDS applies to discard the packets. It is generated at boot time and it is based on the priorities assigned to different flows. According to this policy, packets of the permitted flows are captured and analyzed. The rest of the packets are dropped. Thus, the PDP indicates which action must be carried out for each type of packet.The PDP contains a dynamic list of rules that describe the behavior of the PDS. Every rule has a priority and the discarding policy can be updated dynamically by the Flow_prio Performance Controller System (FPCS).

C. Packet Preprocessing System (PPS)

This module performs small per packet operations that require low CPU utilization. This system is only reached by the packets that the PDP indicates that need to be preprocessed. The PPS treats the packets according to the Packet Preprocessing Policy (PPP). Therefore, it is possible to apply a set of operations to the packets, before they are selected to be analyzed or dropped.

D. Packet Preprocessing Policy (PPP)

The PPP is the decision logic that the PDS uses to carry out the packet preprocessing. Like the PDP, the PPP is subject to be updated throughout the execution of a test. Due to the criticality of the softIRQ, the actions contained in the PPP must be computationally affordable.The PPP indicates which function has to be executed for every flow which requires preprocessing. After that function execution, the packet is captured or dropped.

E. Flow_prio Performance Controller System (FPCS)

This module is responsible for controlling packet capture and analysis according to the probe's state. It runs periodically, and according to the system's state, it modifies the PDP, if it is necessary. When this module is running, it gathers information from the system such as performance statistics, activated flows, priorities, packet queue state, etc.The solution of the Flow_prio mode, is based on the adaptation of the number of captured packets to the available resources for analysis. When the number of packets from the highest priority flows grows, Ksensor may not be able to process all of them. The FPCS detects this situation and configures the system in order to stop capturing packets belonging to flows of lower priority.

If the system is not able to process all the packets because of the growth of the packet rate, it stops capturing lower priority traffic. On the other hand, if the system can process more flows, it starts capturing and analyzing other flows taking into account the priority. The flows that must be processed and their priority are configured at boot time. The new architecture can alsoprocess traffic without taking flows into account. This traffic, called regular traffic, is the lowest priority one. This must be configured at boot time, too.

F. Flow_prio Performance Controller Policy (FPCP)

The FPCP is the policy that the FPCS uses to control the probe. It is loaded at boot time and we can customize it as we do with flows and priorities. This policy decides in which cases the system has to update PDP and which actions should be taken.

G.. Queue Monitoring System (QMS)

This module monitors the analysis queue in order to inform FPCS of its state. QMS is a service routine used by FPCS in order to know the number of packets of the analysis queue at every moment. The FPCS has enough values of the queue state to be able to detect trends and make decisions accordingly.

V. VALIDATION

This section presents performance tests which compare the new Flow_prio mode with the previous running mode of Ksensor. In order to make the performance comparisons, we use a testing architecture [10].

A. Testing Architecture

As Fig. 3 shows, the hardware setup for validation consists of four elements: a traffic generator (injector); a probe (Ksensor) which captures, filters and analyzes traffic; a packet receiver and an element for managing, configuring and launching the tests (manager). All of them are connected to the same Gigabit Ethernet switch.Regarding software, we use the agents, formatters and daemons of the testing architecture and the manager is in charge of its correct operation. It does all the necessary tasks in order to automate the tests and measure the performance metrics of interest. At the end of every test, the manager collects the measurements from Ksensor and the injector.

The basic idea is to overwhelm the system under test, Ksensor, with high traffic generated from the injector. We have installed an Endace 4.3GE DAG card [11] on the injector. This element inputs parameterized traffic into the network. The generated traffic patterns are repeated for the different running modes of Ksensor.We have made tests varying the network packet rate between 50000 and 1500000 packets per second (50-1500 Kpps), with packets of 40 bytes mean length. As not all the packets need to be processed in the analysis stage, a rate called q_a indicates the proportion of the generated

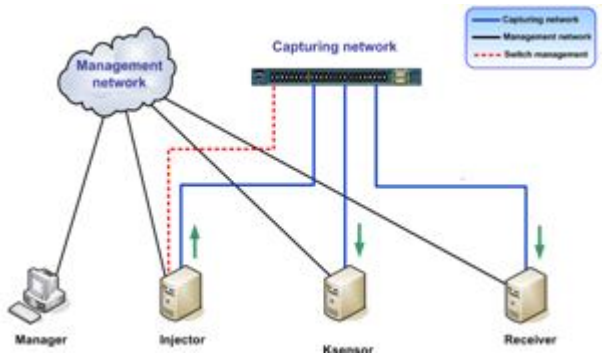

Fig. 3. Hardware setup for validation tests.

DOI: 01.IJRTET.8.1.65

packets that has to be analyzed. Moreover, there are rates q_{ai} with $i \in (1,2,3,\ldots)$ to distinguish the packet proportion of each one of the data flows which we want to filter in Ksensor. The sum of all the q_{ai} is q_a.

In the test scenario, Ksensor is configured with the filtering policies (PDP, PPP and FPCP). In particular, we use a simple PDP which contains 1-4 filtering rules. It is worth mentioning that the generated traffic patterns (with q_{ai} rates) are directly related to the PDP policies. To complete the definition of the filtering policies, we use a basic FPCP and no PPP.

With regard to the analysis phase of Ksensor, in order to test the behavior of the probe on different environments, we use three different simulated analysis loads. To do that, we implement loops that take different number of cycles: 1000 cycles (1K), 5000 cycles (5K) and 25000 cycles (25K).

B. Comparative Tests

We have measured the overall throughput of the probe. This is the total number of analyzed packets per second without taking into account the flow which packets belong to. We have observed that the Flow_prio mode presents a worse result than previous Ksensor's mode. Besides, the bigger is the number of rules the worse is the performance. This is due to the introduction of new features running within the softIRQ.

However, Fig. 4 shows an example comparing the percentage of analyzed packets for each relevant data flow by Ksensor and by the new Ksensor_Flow_prio. We can observe that, while the previous Ksensor treats all the flows in the same way, the Flow_prio mode presents an improvement for the most priority flow (Flow_prio1 in Fig. 4). This is due to the fact that Flow_prio mode uses dynamic filtering and selectively discards packets from the least significant flows, obtaining a better performance for the most relevant data flows. Although Fig. 4 corresponds to a particular case (q_{a1}=0.25, q_{a2}=0.75 and two-rule PDP2 policy in Ksensor_Flow_prio), we have also observed the improvement of the Flow_prio mode in the rest of the tests with the settings previously described.

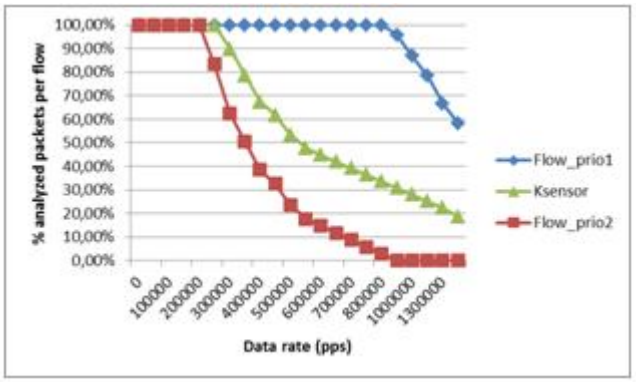

Fig. 4. Percentage of analyzed packets per flow by Ksensor and by Ksensor_Flow_prio with PDP2 (in both scenarios, q_{a1}=0.25, q_{a2}=0.75 and 1K analysis load).

Conclusions

The previous architecture of Ksensor captured and analyzed packets in an efficient way but it did not take flows into account. This paper proposes a new design that makes Ksensor to process flows with priorities. Because of this, the administrator of the probe can select which flows are more relevant and the new discarding system (PDS) assures that the processed packets are those with higher priority. PDS is implemented in the network softIRQ of the operating system. With the performance controller system, the traffic probe can control dynamically which flows are processed and which ones are dropped according to the available resources in the system at any moment. Therefore the new architecture avoids relevant packet losses over irrelevant packet capturing and it can be adapted to different scenarios.Another point of view could have been to consider the implementation of the PDS on a hardware network tap. However, it would not provide the flexibility and modularity that our software solution does.

References

[1] F. Fusco, and L. Deri, "High Speed Network Traffic Analysis with Commodity Multi-core Systems," in Proceedings of Internet Measurement Conference (IMC 2010).Melbourne, Australia, November 2010.

[2] A. Munoz, A. Ferro, F. Liberal, and J. Lopez-Herrera, "Ksensor: Multithreaded Kernel-Level Probe for Passive QoS Monitoring," in IEEE International Conference on Parallel and Distributed Systems (ICPADS 2007). Hsinchu, Taiwan, December 2007.

[3] A. Ferro, I. Delgado, A. Munoz, and F. Liberal, "A Multiprocessor Architecture for Passive Analysis of Network Traffic Focusing on Complex QoS Strategies," in Proceedings of IEEE International Conference on Communications (ICC 2005). Seoul, Korea. May 2005.

[4] L. Deri, J. Gasparakis, P.Waskiewicz, and F. Fusco, "Ksensor: Multithreaded Kernel-Level Probe for Passive QoS Monitoring," in Proceedings of the 1st Workshop on Network-Embedded Management and Applications (NEMA 2010). Niagara Falls, Canada. October 2010.

[5] J. Slaby, "Rapid Data Transfers on COMBO Platform," Diploma Thesis, Brno: Masarykova Univerzita, 2008. http://is.muni.cz/th/98734/fi_m/dp.pdf 15.10.2012

[6] C. Benvenuti, Understanding Linux Network Internals. O'Reilly Media, 2005.

[7] A. Ferro, F. Liberal, A. Munoz, and C. Perfecto, "Network Traffic Sensor for Multiprocessor Architectures: Design Improvement Proposals," in LNCS, vol 1, pp. 146-157. Springer-Verlag, Heidelberg 2004.

[8] A. Ferro, F. Liberal, A. Munoz, I. Delgado, and A. Beaumont, "Software Architecture Based on Multiprocessor Platform to Apply Complex Intrusion Detection Techniques," in IEEE Conference on Security Technology. Las Palmas, Spain, 2005.

[9] R. Love, Linux Kernel Development, 3rd Edition. Adisson Wesley, 2010.

[10] A. Pineda, L. Zabala, and A. Ferro, "Network Architecture to Automatically Test Traffic Monitoring Systems," in Mosharaka International Conference on Communications & Signal Processing. Barcelona, Spain, April 2012.

[11] Endace DAG Cards Enterprise Network Monitoring Tools. http://www.endace.com 15.10.2012.

DOI: 01.IJRTET.8.1.65